

Exhibit 14 to Complaint
Intellectual Ventures I LLC and Intellectual Ventures II LLC

**Example American Count 6 Systems and Services
U.S. Patent No. 7,257,582 (“’582 Patent”)**

The Accused Systems and Services include without limitation American systems and services that utilize Spark; all past, current, and future systems and services that operate in the same or substantially similar manner as the specifically identified systems and services; and all past, current, and future American systems and services that have the same or substantially similar features as the specifically identified systems and services (“Example American Count 6 Systems and Services” or “American Systems and Services”).¹

On information and belief, the American Systems and Services use Spark in its private cloud(s). For example, American posts, or has posted, job opportunities that require familiarity with Spark technology.

- Example of a Senior Java Full Stack Developer at American Airlines whose position mentions data transformation and streaming development via Spark. <https://www.linkedin.com/in/rohitha-m6363/> (Last accessed on 9/19/24)
- Example of a Big Data Engineer at American Airlines whose position mentions the use of Spark. <https://www.linkedin.com/in/krishna-karthika-katragadda-149786306/> (Last accessed on 9/19/24)
- Example of a Senior Data Engineer at American Airlines whose position mentions analyzing and developing programs, writing programs, and developing jobs via Spark. <https://www.linkedin.com/in/sriram1993/> (Last accessed on 9/19/24)
- Example of a Senior Data Engineer at American Airlines whose position mentions the use of Spark. <https://www.linkedin.com/in/madhu-sudhan-reddy-y-b3897b129/> (Last accessed on 9/19/24)
- Example of a Data Engineer at American Airlines whose position mentions the use of Spark. <https://www.linkedin.com/in/ashish-n-013372287/> (Last accessed on 9/19/24)
- Example of Analyst/Sr Analyst, Safety Data Analytics Job Listing at American Airlines which mentions use of Spark. https://jobs.aa.com/job/AnalystSr-Analyst%2C-Safety-Data-Analytics/74823-en_US (Last accessed on 9/19/24)

As another example, American has announced cloud migration of legacy technology and efforts to modernize its mainframes and servers. Source: <https://dxc.com/sg/en/insights/customer-stories/american-airlines-cloud-data-automation>.

On information and belief, other information confirms American uses Spark technology.

¹ For the avoidance of doubt, Plaintiffs do not accuse public clouds of American if those services are provided by a cloud provider with a license to Plaintiffs’ patents that covers American’s activities. IV will provide relevant license agreements for cloud providers in discovery. To the extent any of these licenses are relevant to American’s activities, Plaintiffs will meet and confer with American about the impact of such license(s).

American Airlines

American Airlines' purpose is to care for people on life's journey, and its commitment to delivering exceptional service and operational reliability is unwavering. Leveraging the Databricks Data Intelligence Platform, American aimed to reduce the number of mishandled bags during transfers by analyzing datasets and automating the baggage handling process. The result was an impressive improvement in processing performance, significantly reducing the number of misplaced bags and enhancing the efficiency of baggage handling.

This improvement was made possible by the Next-Generation Stream Processing Engine, Spark Structured Streaming, which powers data streaming on the Databricks Platform. This unified API for batch and stream processing enabled American to centralize data ingestion and facilitate real-time data gathering and reporting — a testament to the power of leveraging technology to drive operational efficiency and deliver exceptional service. American also sought to optimize staffing efficiencies by enhancing operational processes overseen by assigned baggage-running allocators. The Databricks Data Intelligence Platform was utilized to streamline data ingestion, storage, and access — enhancing analysis and reporting, reducing manual interventions, and improving productivity.

Source: <https://www.databricks.com/blog/disrupting-status-quo-through-data-and-ai-celebrating-2024-data-team-disruptor-award-nominees>.²

This article describes how Apache Spark is related to Databricks and the Databricks Data Intelligence Platform.

Apache Spark is at the heart of the Databricks platform and is the technology powering compute clusters and SQL warehouses. Databricks is an optimized platform for Apache Spark, providing an efficient and simple platform for running Apache Spark workloads.

² All sources cited in this document were publicly accessible as of the filing date of the Complaint.

Source: <https://docs.databricks.com/en/spark/index.html>.

Sponsored by: Striim | Powering a Delightful Travel Experience with a Real-Time Operational Data Hub



Databricks
117K subscribers

Subscribe



3



Share



Save



409 views Jul 26, 2023 [SAN FRANCISCO](#)

American Airlines champions operational excellence in airline operations to provide the most delightful experience to our customers with on-time flights and meticulously maintained aircraft. To modernize and scale technical operations with real-time, data-driven processes, we delivered a DataHub that connects data from multiple sources and delivers it to analytics engines and systems of engagement in real-time. This enables operational teams to use any kind of aircraft data from almost any source imaginable and turn it into meaningful and actionable insights with speed and ease. This empowers maintenance hubs to choose the best service and determine the most effective ways to utilize resources that can impact maintenance outcomes and costs. The end-product is a smooth and scalable operation that results in a better experience for travelers. In this session, you will learn how we combine an operational data store (MongoDB) and a fully managed streaming engine (Striim) to enable analytics teams using Databricks with real-time operational data.

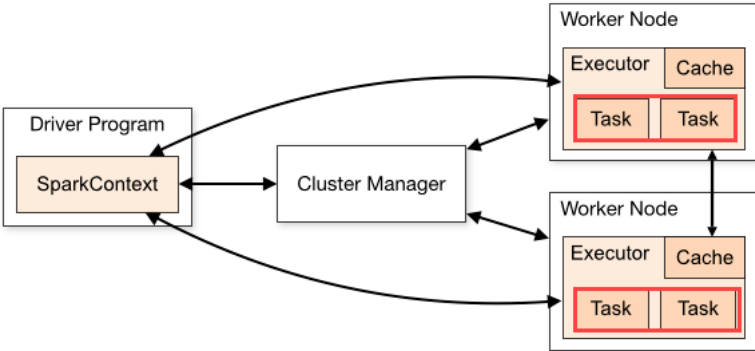
Source: https://www.youtube.com/watch?v=NmdBiO_U7rA.

The thumbnail features a dark blue background with white text. The main title 'Optimizing Spark and PySpark for Improved Big Data Performance' is at the top. Below it is a subtitle 'Continuous and autonomous optimization for Spark and PySpark workloads empowering more efficient data engineering, data science and machine learning'. At the bottom, there is a row of logos for Amazon EMR, Cloudera, Hadoop, HDInsight, and Databricks.


Optimizing Spark and PySpark for Improved Big Data Performance


Continuous and autonomous optimization for Spark and PySpark workloads empowering more efficient data engineering, data science and machine learning


Source: <https://granulate.io/solutions/spark/>.

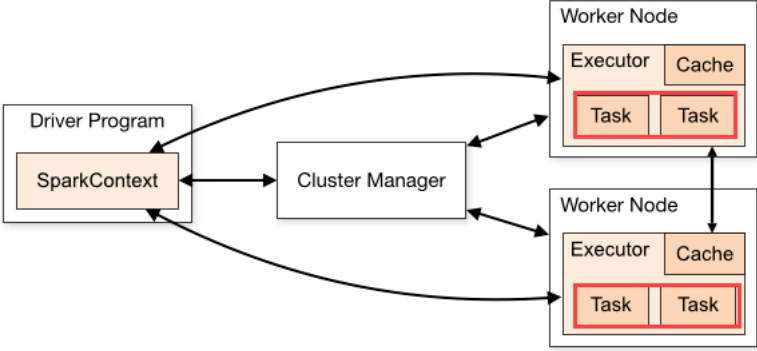
| U.S. Patent No. 7,257,582 (Claim 1) | |
|--|---|
| Claim(s) | Example American Count 6 Systems and Services |
| 1. A method of effecting on a preexisting input file a computer-executable process comprised of a plurality of subtasks, the method comprising the steps of: | <p>To the extent this preamble is limiting, on information and belief, the American Count 6 Systems and Services practice a method of effecting on a preexisting input file a computer-executable process comprised of a plurality of subtasks.</p> <p>Spark includes a Cluster Mode where it connects to multiple cluster managers that allocate resources across applications. For example, Spark uses executors on nodes in the clusters to run tasks corresponding to an input file/data set that has been transformed into a resilient distributed dataset (RDD).</p> <h3>Components</h3> <p>Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in your main program (called the <i>driver program</i>).</p> <p>Specifically, to run on a cluster, the SparkContext can connect to several types of <i>cluster managers</i> (either Spark's own standalone cluster manager, Mesos, YARN or Kubernetes), which allocate resources across applications. Once connected, Spark acquires <i>executors</i> on nodes in the cluster, which are processes that run computations and store data for your application. Next, it sends your application code (defined by JAR or Python files passed to SparkContext) to the executors. Finally, SparkContext sends <i>tasks</i> to the executors to run.</p> <p>Source: https://spark.apache.org/docs/latest/cluster-overview.html.³</p>  <pre> graph LR subgraph DP [Driver Program] SC[SparkContext] end CM[Cluster Manager] subgraph WN1 [Worker Node] E1[Executor] C1[Cache] T1[Task] E1 --- C1 E1 --- T1 end subgraph WN2 [Worker Node] E2[Executor] C2[Cache] T2[Task] E2 --- C2 E2 --- T2 end SC <--> CM SC --> WN1 SC --> WN2 CM --> WN1 CM --> WN2 E1 --> E2 E2 --> E1 </pre> |

³ Annotations added unless otherwise noted.

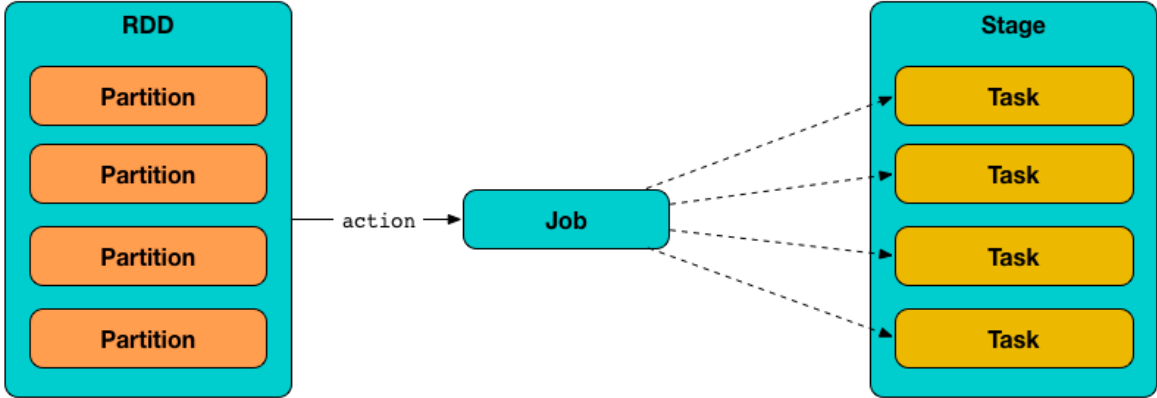
| U.S. Patent No. 7,257,582 (Claim 1) | |
|--|--|
| Claim(s) | Example American Count 6 Systems and Services |
| | <p>Source: https://spark.apache.org/docs/latest/cluster-overview.html.</p> <p>Overview </p> <p>At a high level, every Spark application consists of a <i>driver program</i> that runs the user's main function and executes various <i>parallel operations</i> on a cluster. The main abstraction Spark provides is a <i>resilient distributed dataset</i> (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to <i>persist</i> an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures.</p> <p>Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.</p> <p>Resilient Distributed Datasets (RDDs)</p> <p>Spark revolves around the concept of a <i>resilient distributed dataset</i> (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. There are two ways to create RDDs: <i>parallelizing</i> an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.</p> <p>Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.</p> |
| (a) automatically determining file allocation and logically subdividing records of said input file into a plurality of partitions; | <p>On information and belief, the American Count 6 Systems and Services practice automatically determining file allocation and logically subdividing records of said input file into a plurality of partitions.</p> <p>Spark creates an RDD from datasets including files. This process includes forming a logical division of the datasets. An RDD results in an automatic file allocation across multiple nodes in a cluster, so that the nodes of the cluster can operate on the RDD in parallel.</p> |

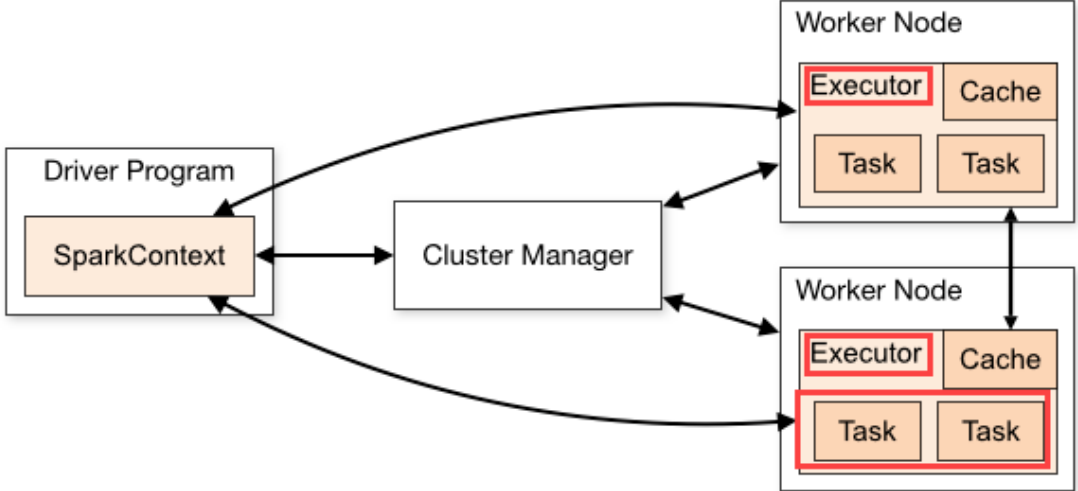
| U.S. Patent No. 7,257,582 (Claim 1) | |
|--|--|
| Claim(s) | Example American Count 6 Systems and Services |
| | <p>Resilient Distributed Datasets (RDDs)</p> <p>Spark revolves around the concept of a <i>resilient distributed dataset</i> (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. There are two ways to create RDDs: <i>parallelizing</i> an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.</p> <p>Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.</p> <p>Overview </p> <p>At a high level, every Spark application consists of a <i>driver program</i> that runs the user's main function and executes various <i>parallel operations</i> on a cluster. The main abstraction Spark provides is a <i>resilient distributed dataset</i> (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to <i>persist</i> an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures.</p> <p>Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.</p> <p>One important parameter for parallel collections is the number of <i>partitions</i> to cut the dataset into. Spark will run one task for each partition of the cluster. Typically you want 2-4 partitions for each CPU in your cluster. Normally, Spark tries to set the number of partitions automatically based on your cluster. However, you can also set it manually by passing it as a second parameter to <code>parallelize</code> (e.g. <code>sc.parallelize(data, 10)</code>). Note: some places in the code use the term <i>slices</i> (a synonym for partitions) to maintain backward compatibility.</p> <p>Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.</p> |
| (b) distributing descriptions of all of said partitions to each of a plurality of subtask processors | <p>On information and belief, the American Count 6 Systems and Services practice distributing descriptions of all of said partitions to each of a plurality of subtask processors.</p> <p>Spark distributes partitions to clusters which includes multiple machines or nodes for processing. For example, Spark prepares jobs to perform work on the partitions. Jobs are divided into stages, and stages are divided into tasks, with one task assigned for each partition.</p> |

| U.S. Patent No. 7,257,582 (Claim 1) | |
|--|--|
| Claim(s) | Example American Count 6 Systems and Services |
| | <p>Overview </p> <p>At a high level, every Spark application consists of a <i>driver program</i> that runs the user's main function and executes various <i>parallel operations</i> on a cluster. The main abstraction Spark provides is a <i>resilient distributed dataset</i> (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to <i>persist</i> an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures.</p> <p>Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.</p> <p>One important parameter for parallel collections is the number of <i>partitions</i> to cut the dataset into. Spark will run one task for each partition of the cluster. Typically you want 2-4 partitions for each CPU in your cluster. Normally, Spark tries to set the number of partitions automatically based on your cluster. However, you can also set it manually by passing it as a second parameter to <code>parallelize</code> (e.g. <code>sc.parallelize(data, 10)</code>). Note: some places in the code use the term <i>slices</i> (a synonym for partitions) to maintain backward compatibility.</p> <p>Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.</p> <p>Apache Spark's Resilient Distributed Datasets (RDD) are a collection of various data that are so big in size, that they cannot fit into a single node and should be partitioned across various nodes. Apache Spark automatically partitions RDDs and distributes the partitions across different nodes. They are evaluated lazily (i.e, the execution will not start until an action is triggered which increases manageability, saves computation and thus increases optimization and speed) and the transformations are stored as <i>directed acyclic graphs</i> (DAG). So, every action on the RDD will make Apache Spark recompute the DAG.</p> <p>Source: https://www.talend.com/resources/intro-apache-spark-partitioning/.</p> |
| (c) simultaneously executing at least a respective one of the subtasks of the computer-executable process in each of at least some of said | <p>On information and belief, the American Count 6 Systems and Services practice simultaneously executing at least a respective one of the subtasks of the computer-executable process in each of at least some of said processors on a respective one of the partitions with each subtask reading and processing the respective partition so as to process the respective partition and produce respective subtask output.</p> |

| U.S. Patent No. 7,257,582 (Claim 1) | |
|---|--|
| Claim(s) | Example American Count 6 Systems and Services |
| processors on a respective one of the partitions with each subtask reading and processing the respective partition so as to process the respective partition and produce respective subtask output and; | <p>Spark worker nodes simultaneously execute at least one task relating to the partition.</p> <h3>Components</h3> <p>Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in your main program (called the <i>driver program</i>).</p> <p>Specifically, to run on a cluster, the SparkContext can connect to several types of <i>cluster managers</i> (either Spark's own standalone cluster manager, Mesos, YARN or Kubernetes), which allocate resources across applications. Once connected, Spark acquires <i>executors</i> on nodes in the cluster, which are processes that run computations and store data for your application. Next, it sends your application code (defined by JAR or Python files passed to SparkContext) to the executors. Finally, SparkContext sends tasks to the executors to run.</p> <p>Source: https://spark.apache.org/docs/latest/cluster-overview.html.⁴</p>  <p>The diagram illustrates the Spark architecture. On the left, a box labeled 'Driver Program' contains a sub-box labeled 'SparkContext'. In the center is a box labeled 'Cluster Manager'. On the right are two 'Worker Node' boxes. Each Worker Node contains an 'Executor' box and a 'Cache' box. Inside each 'Executor' box are two 'Task' boxes. Arrows show the following connections: a double-headed arrow between 'SparkContext' and 'Cluster Manager'; a double-headed arrow between 'SparkContext' and each 'Worker Node'; and a double-headed arrow between each 'Cluster Manager' and each 'Worker Node'.</p> <p>Source: https://spark.apache.org/docs/latest/cluster-overview.html.</p> |

⁴ Annotations added unless otherwise noted.

| U.S. Patent No. 7,257,582 (Claim 1) | |
|-------------------------------------|--|
| Claim(s) | Example American Count 6 Systems and Services |
| | <p>In Spark, each stage is built from transformations that can be done in a row, without the need for shuffle (narrow transformations). To recap, stages are created based on chunks of processing that can be done in a parallel manner, without shuffling things around again.</p>  <p>Source: https://engineering.salesforce.com/how-to-optimize-your-apache-spark-application-with-partitions-257f2c1bb414/.</p> <p>Spark's tasks relating to each partition are being executed in parallel by the worker nodes.</p> <p>There are several useful things to note about this architecture:</p> <ol style="list-style-type: none"> 1. Each application gets its own executor processes, which stay up for the duration of the whole application and run tasks in multiple threads. This has the benefit of isolating applications from each other, on both the scheduling side (each driver schedules its own tasks) and executor side (tasks from different applications run in different JVMs). However, it also means that data cannot be shared across different Spark applications (instances of SparkContext) without writing it to an external storage system. <p>Source: https://spark.apache.org/docs/latest/cluster-overview.html.</p> |

| U.S. Patent No. 7,257,582 (Claim 1) | |
|-------------------------------------|---|
| Claim(s) | Example American Count 6 Systems and Services |
| |  <p>The diagram illustrates the Spark architecture. On the left, a box labeled 'Driver Program' contains a sub-box 'SparkContext'. In the center is a box labeled 'Cluster Manager'. On the right are two 'Worker Node' boxes. Each Worker Node contains an 'Executor' (highlighted with a red border), a 'Cache', and two 'Task' boxes (the bottom 'Task' box in each node is also highlighted with a red border). Arrows show bidirectional communication between 'SparkContext' and the 'Cluster Manager', and between the 'Cluster Manager' and each 'Worker Node'. Additionally, curved arrows point from 'SparkContext' directly to each 'Worker Node'.</p> <p>In Spark, data is generally not distributed across partitions to be in the necessary place for a specific operation. During computations, a single task will operate on a single partition – thus, to organize all the data for a single <code>reduceByKey</code> reduce task to execute, Spark needs to perform an all-to-all operation. It must read from all partitions to find all the values for all keys, and then bring together values across partitions to compute the final result for each key – this is called the shuffle.</p> <p>Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.</p> <p>The Shuffle is an expensive operation since it involves disk I/O, data serialization, and network I/O. To organize data for the shuffle, Spark generates sets of tasks – <i>map</i> tasks to organize the data, and a set of <i>reduce</i> tasks to aggregate it. This nomenclature comes from MapReduce and does not directly relate to Spark's map and reduce operations.</p> <p>Internally, results from individual map tasks are kept in memory until they can't fit. Then, these are sorted based on the target partition and written to a single file. On the reduce side, tasks read the relevant sorted blocks.</p> <p>Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.</p> |

| U.S. Patent No. 7,257,582 (Claim 1) | |
|--|--|
| Claim(s) | Example American Count 6 Systems and Services |
| (d) thereafter repeating step (c) in at least some of the subtask processors each with another unprocessed partition on a first-come/first-served basis; and | <p>On information and belief, the American Count 6 Systems and Services practice repeating step (c) in at least some of the subtask processors each with another unprocessed partition on a first-come/first-served basis.</p> <p>Spark task output produced by each worker node is combined to produce processed output for the related partition.</p> <p>The simplest option, available on all cluster managers, is <i>static partitioning</i> of resources. With this approach, each application is given a maximum amount of resources it can use and holds onto them for its whole duration. This is the approach used in Spark's <i>standalone</i> and <i>YARN</i> modes, as well as the <i>coarse-grained Mesos mode</i>. Resource allocation can be configured as follows, based on the cluster type:</p> <ul style="list-style-type: none"> • Standalone mode: By default, applications submitted to the standalone mode cluster will run in FIFO (first-in-first-out) order, and each application will try to use all available nodes. You can limit the number of nodes an application uses by setting the <code>spark.cores.max</code> configuration property in it, or change the default for applications that don't set this setting through <code>spark.deploy.defaultCores</code>. Finally, in addition to controlling cores, each application's <code>spark.executor.memory</code> setting controls its memory use. <p>Source: https://spark.apache.org/docs/latest/job-scheduling.html.</p> <p>1 Available resources in your cluster</p> <p>Spark's official recommendation is that you have ~3x the number of partitions than available cores in cluster, to maximize parallelism along side with the overhead of spinning more executors. But this is not quite so straight forward. If our tasks are rather simple (taking less than a second), we might want to consider decreasing our partitions (to avoid the overhead), even if it means less than 3x. We should also take under consideration the memory of each executor, and make sure we are not exceeding it. If we do, we might want to create more partitions, even if it's more than 3x our available cores, so that each will be smaller, or increase the memory of our executors.</p> <p>Source: https://engineering.salesforce.com/how-to-optimize-your-apache-spark-application-with-partitions-257f2c1bb414/.</p> |

| U.S. Patent No. 7,257,582 (Claim 1) | |
|--|---|
| Claim(s) | Example American Count 6 Systems and Services |
| | <p>The diagram illustrates the Spark architecture. On the left, a box labeled 'Driver Program' contains a sub-box 'SparkContext'. In the center is a box labeled 'Cluster Manager'. On the right are two 'Worker Node' boxes. Each Worker Node contains an 'Executor' (highlighted with a red border), a 'Cache', and two 'Task' boxes. Arrows show bidirectional communication between 'SparkContext' and 'Cluster Manager', and between 'Cluster Manager' and each 'Worker Node'. Additionally, curved arrows point from 'SparkContext' directly to the 'Executor' in each 'Worker Node'.</p> <p>Source: https://spark.apache.org/docs/latest/cluster-overview.html.</p> <p>By default, Spark's scheduler runs jobs in FIFO fashion. Each job is divided into "stages" (e.g. map and reduce phases), and the first job gets priority on all available resources while its stages have tasks to launch, then the second job gets priority, etc. If the jobs at the head of the queue don't need to use the whole cluster, later jobs can start to run right away, but if the jobs at the head of the queue are large, then later jobs may be delayed significantly.</p> <p>Source: https://spark.apache.org/docs/latest/job-scheduling.html.</p> |
| (e) generating at least one output combining all of the subtask outputs and reflecting the processing of all of said subtasks. | <p>On information and belief, the American Count 6 Systems and Services practice generating at least one output combining all of the subtask outputs and reflecting the processing of all of said subtasks.</p> <p>Spark task output produced by each worker node is combined to produce final processed output for the corresponding partition.</p> |

| U.S. Patent No. 7,257,582 (Claim 1) | |
|-------------------------------------|---|
| Claim(s) | Example American Count 6 Systems and Services |
| | <p>In Spark, data is generally not distributed across partitions to be in the necessary place for a specific operation. During computations, a single task will operate on a single partition – thus, to organize all the data for a single <code>reduceByKey</code> reduce task to execute, Spark needs to perform an all-to-all operation. It must read from all partitions to find all the values for all keys, and then bring together values across partitions to compute the final result for each key – this is called the shuffle.</p> <p>Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.</p> <p>The Shuffle is an expensive operation since it involves disk I/O, data serialization, and network I/O. To organize data for the shuffle, Spark generates sets of tasks – <i>map</i> tasks to organize the data, and a set of <i>reduce</i> tasks to aggregate it. This nomenclature comes from MapReduce and does not directly relate to Spark's <code>map</code> and <code>reduce</code> operations.</p> <p>Internally, results from individual map tasks are kept in memory until they can't fit. Then, these are sorted based on the target partition and written to a single file. On the reduce side, tasks read the relevant sorted blocks.</p> <p>Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.</p> |